IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2019, No. 4, pp. 272–289. DOI:10.13154/tosc.v2019.i4.272-289

Weak Keys in the Rekeying Paradigm: Application to COMET and mixFeed

Mustafa Khairallah

School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore, Singapore
mustafam001@e.ntu.edu.sg

Abstract. In this paper, we study a group of AEAD schemes that use rekeying as a technique to increase efficiency by reducing the state size of the algorithm. We provide a unified model to study the behavior of the keys used in these schemes, called Rekey-and-Chain (RaC). This model helps understand the design of several AEAD schemes. We show generic attacks on these schemes based on the existence of certain types of weak keys. We also show that the borderline between multi-key and single-key analyses of these schemes is not solid and the analysis can be performed independent of the master key, leading sometimes to practical attacks in the multi-key setting. More importantly, the multi-key analysis can be applied in the single key setting, since each message is encrypted with a different key. Consequently, we show gaps in the security analysis of COMET and mixFeed in the single key setting, which led the designers to provide overly optimistic security claims. In the case of COMET, full key recovery can be performed with 2^{64} online queries and 2^{64} offline queries in the single-key setting, or 2^{46} online queries per user and 2^{64} offline queries in the multi-key setting with ~ 0.5 million users. In the case of mixFeed, we enhance the forgery adversarial advantage in the single-key setting with a factor of 2^{67} compared to what the designers claim. More importantly, our result is just a lower bound of this advantage, since we show that the gap in the analysis of mixFeed depends on properties of the AES Key Schedule that are not well understood and require more cryptanalytic efforts to find a more tight advantage. After reporting these findings, the designers updated their security analyses and accommodated the proposed attacks.

Keywords: weak keys · authenticated encryption · comet · mixfeed · nist · forgery · AEAD

1 Introduction

Lightweight Symmetric Key Cryptography has been a growing research area in the past 10 years or more, with applications varying from block cipher design to authenticated encryption or hash functions and much more. This has lead the National Institute of Standards and Technology (NIST) to release a call for proposals for a new lightweight cryptography standard [nis18], to be used in applications such as Internet-of-Things (IoT) and Sensor Networks. Authenticated Encryption with Associated Data (AEAD) is one of the most important requirements of symmetric key cryptography (SKE) in these environments, since it is usually cheaper than having independent solutions for the authentication and encryption requirements of the system. Most of the lightweight AEAD designs fall into one of three families: (Tweakable) Block-Cipher Based schemes, Permutation-Based schemes and Stream-Cipher Based schemes, with the last two families sometimes overlapping.

Received: 2019-09-01, Accepted: 2019-11-01, Published: 2020-01-31

Table 1: Some lightweight AEAD designs and the techniques used in them.

Mode	Chaining	Rekeying	Masking	Assumption
Romulus-N1 [IKMP19c, IKMP19a]	✓	-	-	TPRP
COFB [CIMN17, BCI+19]	✓	-	✓	PRP
HyENA [CDJN19]	✓	-	✓	PRP
Remus-N1 [IKMP19b, IKMP19a]	✓	✓	-	ICM
Remus-N2 [IKMP19b, IKMP19a]	✓	✓	✓	ICM
COMET-128 [GJN19a]	✓	✓	-	ICM
mixFeed [CN19a]	✓	✓	-	ICM

One of the main challenges facing the designers of AEAD schemes is reducing the state size of the algorithm. The state size translates either into memory limitations for software implementations or hardware area for ASIC/FPGA implementations. Most of the time the state size is the main contributor to how small the implementation of an algorithm can be. For example, the low area implementation of the SKINNY Lightweight Block Cipher [BJK+16] with 128-bit block and 128-bit key is only $\sim 23\%$ smaller that the corresponding round-based implementation. In order to design AEAD schemes based on block ciphers that have limited state size while meeting certain security goals, two techniques have recently become popular. The first one is chaining, where the input to a block cipher is a linear combination of the previous output of the block cipher and the current input block. This ensures that the input to a block cipher depends on the previous block cipher calls and not just the current input block. The second technique is Rekeying, where at the beginning of the encryption process, a random value is generated based on the master key, and this value is later used as during the block cipher calls, generally as a tweaked key. In addition to helping reduce the state size, rekeying is also useful against physical attacks, such as Side Channel and Fault Attacks, by limiting the exposure/usage of the master key. Another technique that is similar to Rekeying, is masking, where a random value is initially generated based on the master key and a unique *nonce*, then it is used in a construction on top of the block cipher to increase the randomness.

While the latter two techniques are relatively similar, a fundamental difference is that rekeying usually requires stronger assumptions about the underlying Block Cipher as the security of the mode can only be proven in the Ideal-Cipher Model (ICM) as opposed to the (Tweakable) Pseudo-Random Permutation Model ((T)PRP). Table 1 shows some of the current AEAD designs and which of these three techniques they use. The comparison is limited to (online) rate-1 modes that are secure in the nonce respecting model and use chaining, i.e. modes that use almost a single block cipher call per input block and are secure in the nonce respecting model. Table 1 does not serve as an exhaustive list. The modes in Table 1 represent around 25% of the block-cipher based submissions to the NIST lightweight cryptography standardization process and and 10% of the overall designs. Moreover, they can be considered as part of the state-of-the-art of this field. Hence, the security analyses and claims made by the designers have to be carefully assessed and scrutinized by the community in order to have to understand them and have a higher trust of their security.

Contributions In this paper, we study the problem of Weak Keys in Rekeying/Masking-Based schemes. We give a framework to model and analyze weak keys in chained block-cipher based modes. We show that, especially for authenticity, the distinction between the single-key and multi-key settings can be blurred and the existence of weak keys can lead to security degradation if they are not handled properly.

We apply this framework to five round 1 AEAD Candidates for the NIST Lightweight Cryptography Process: GIFT-COFB, HyENA, COMET, Remus and mixFeed. We show that there are gaps in the initial analyses provided by the designers of COMET and mixFeed

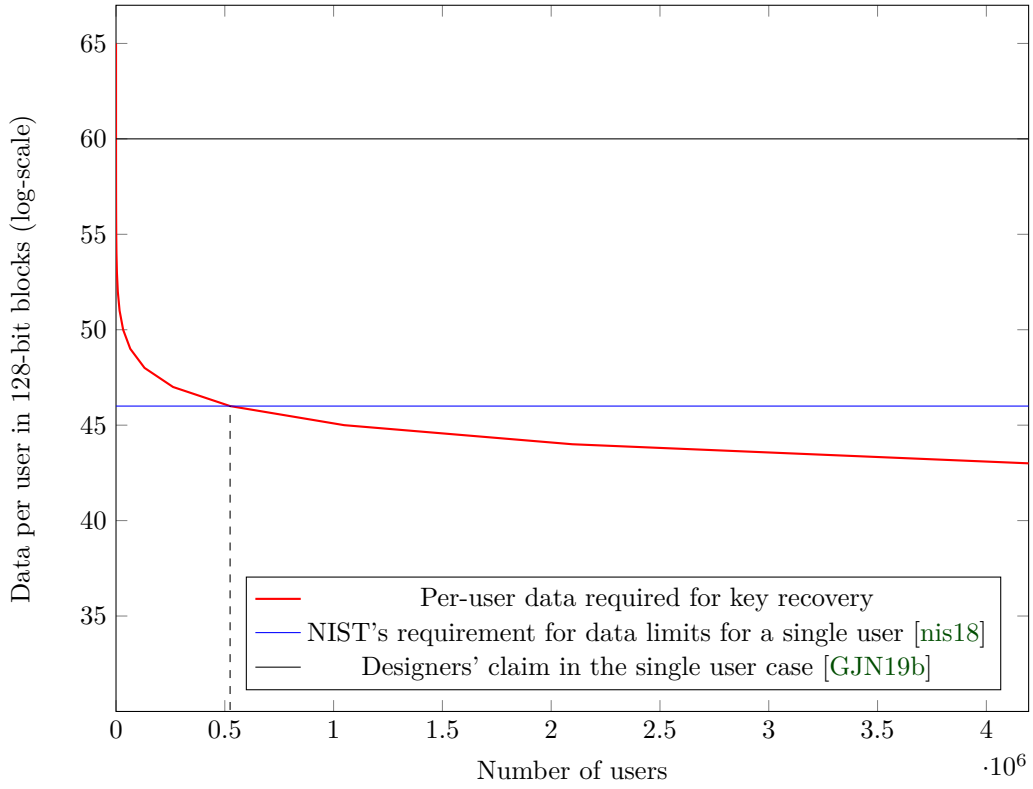

Figure 1: Bit security degradation vs. the number of users for COMET-128

in round 1, leading to overly optimistic claims. Our attacks still fall outside the security requirements in the call for submissions of the NIST and hence does not threaten the security within these limits. However, the results are worrisome in the multi-key setting, where many users are using the scheme and can be targeted by the same attacker, which is a realistic model in scenarios such as IoT and ubiquitous computing. For example, in this model, we show that the master key of at least one of the users of COMET-128 can be recovered with only $\mathcal{O}(2^{64}/\mu)$ online queries per user and 2^{65} offline queries by targeting μ users. This means that if only ~ 0.5 million users adopt this algorithm, the schemes is vulnerable to a key recovery attack on one of the users, even if all the users respect the data limits required by the NIST, as shown in Figure 1. To put it differently, the NIST requires designs to be secure as long as the data complexity is $< 2^{50}$ bytes and the time complexity is $< 2^{112}$. For COMET, in order to operate within these bounds, the number of users must not exceed ~ 0.5 million users, with key recovery possible as soon as this limit is reached. The number of expected vulnerable keys also grows with the number of users, with a factor of 2^{-19}, i.e. e.g. ~ 4 million users respecting the data limit of 2^{50} bytes, it is expected that 8 users are vulnerable. While it is expected that security drops when increasing the number of users, we believe the data limits on each individual users should not drop, as a user does not know if he is being targeted by a single-user or multi-user adversary. Besides, this security drop is not true or similar for all schemes, as proven by Luykx et al. in [LMP17], and 0.5 million users is a very small security margin.

Besides, we show that in the case of mixFeed there is a lack of understanding of the underlying primitive that will require extra assumptions in order to prove the security of the scheme even with bit security as low as 50. To the best of our knowledge, our results represent the first cryptanalytic result for COMET and the first result for mixFeed

in the nonce-respecting model. The designers of mixFeed initially conjectured that the design provides integrity nonce-misuse resistance up to a data complexity of 2^{32}, which was disproven by an attack provided by Khairallah [Kha19], requiring a single repetition of the nonce and a negligible amount of data. In the case of COMET, we give two potential fixes, one with higher performance/area cost, yet more secure, and one with virtually no additional cost and acceptable security.

Note We have communicated our results in the single-key setting to the designers of COMET and mixFeed early on, and in both cases they have verified our analysis. In the case of mixFeed, the designers proposed adding a extra assumption in order to prove the security [CN19b]. In the case of COMET the designers proposed an updated proof that captures our attack [GJN19b], arguing that while the attack needs to be taken into consideration, it does not change their claims. While we believe a fair reading of the specification does not lead to the conclusion that our attack breaches the data bounds of the scheme, we understand that sometimes authors intend to say something that gets misunderstood by the reader and sometimes writing is prone to mistakes. Our goal is not to show whether the data complexity claims are valid or not, but to show the gaps in the analysis provided by the authors, in order to have better understanding of the proposals. Hence, we only focus on the security bounds provided by the designers and not the simplified complexity tables.

2 Background and Motivation

2.1 Weak Key Analysis of Authenticated Encryption

Weak keys are defined as keys that behave in a non-expected manner compared to how the encryption algorithm is intended and detecting whether a secret key belongs to the set of weak keys is easy [HP08]. For example, if an algorithm requires that every call to the primitive used in that algorithm uses a different key, but for some keys the key is fixed for all primitive calls, that is an unexpected behavior. In many cases (as the cases we target in this article), this behavior is detectable with one verification query, since if the weak key occurs, it immediately leads to forgery. Usually, the security bounds are calculated on average over the whole key space, which limits the vulnerability of the weak keys in the single key setting.

Whether the weak keys are exploitable or not is a different problem. An example of a case where the weak keys were exploitable in the AEAD forgery context is the attack on POET by Abdelraheem et al. [ABT14].

2.2 Multi-Key Analysis

In [LMP17], Luykx et al. showed that the effect of analyzing the security of a symmetric key primitive in the multi-key setting can lead to drastic security degradation when the same primitive is used by a huge number of users using different keys. They gave a formula for the adversarial advantage gain against a scheme that has weak keys when used by many users at the same time. If the probability that any key is a weak key is p, the probability that all the μ keys (where μ is the number of users) are not weak is given by

$$(1-p)^{\mu} \tag{1}$$

and the probability that at least one of them is weak is given by

$$1-(1-p)^{\mu} \tag{2}$$

This probability increases almost linearly when μ is small and approaches 1 when $\mu > p^{-1}$. For example, if $p = 2^{-32}$, the advantage is 0.98 at $\mu = 2^{34}$, 0.63 at $\mu = 2^{32}$, 0.39 at $\mu = 2^{31}$, 2^{-16} at $\mu = 2^{16}$ and 2^{-31} at $\mu = 2$. In [BT16], Bellare and Tackmann analyzed GCM with respect to multi-key security degradation, establishing security bounds with a factor μ, showing that the success probability of the adversary can be bounded by $\frac{\mu\sigma^2}{2^n}$, where σ is the overall number of resources available to the adversary. This bound is troublesome since it shows that a multi-key adversary may be able to pay less resources per key compared to a single-key adversary. Specifically, if such bound was tight, the per-key cost would decrease with a factor of $\mu^{-3/2}$ in the multi-key setting. Fortunately, Luykx et al. [LMP17] showed that this bound is not tight and that the multi-key security of GCM does not depend on μ once the underlying cipher is replaced with a uniformly distributed random permutation. More importantly, they provided a sufficient and provable/falsifiable condition for which a scheme would enjoy no security degradation in the multi-key setting. Hence, it is only natural that we need to study new schemes against this condition and show which schemes are insecure in the multi-key setting.

NIST candidates for the new lightweight cryptography standard has been required so far to be secure only in the single key setting [nis18]. Usually, considering only the single key setting reduces the vulnerabilities related to weak keys compared to the multi-key setting [LMP17]. However, the multi-key analysis is relevant to our paper in two ways:

1. Modes such as COMET, mixFeed, COFB, HyENA and Remus use rekeying as a technique even in the single key setting. Hence, we show that the single key cryptanalysis depends on the multi-key setting.

2. Even if it is not required for the standardization process, understanding and analyzing new schemes in different models that capture realistic world views is a crucial task, given the drastic impact that these new schemes will have if they are actually adopted by the industry.

3 COFB-like Schemes

The COmbined FeedBack mode (COFB) [CIMN17] is a popular lightweight block-cipher based AEAD mode that was proposed by Chakraborti et al. at CHES 2017. It is the basis of the GIFT-COFB NIST candidate [BCI+19] and several other proposals have tried to address the shortcomings of COFB by proposing closely related, yet different designs, such as: Romulus [IKMP19c], Remus [IKMP19b], HyENA [CDJN19], COMET [GJN19a] and mixFeed [CN19a]. As part of the security proof of COFB, the authors introduced the idealized COFB (iCOFB) mode, and most of the examples we mentioned earlier can be viewed as different instantiations of (iCOFB), with some changes to the linear function ρ. While it may be intriguing to use the same representation of iCOFB to analyze these modes, it is not the most adequate representation to capture our analysis. Instead, we propose a new representation, which we call Rekey-and-Chain (RaC).

Notation E is a block cipher with key K, $\tilde{E}$ is a tweakable block cipher with tweakey Z. N is the nonce which must be used only once. K is the master key. Z is the message tweakey. IV is the initial value after rekeying. A, M, C are the associated data, message and ciphertext strings, respectively. The underlying cipher has block size n, key size κ. `Perm` is a permutation defined over the tweakey space. i.e. $\mathbb{F}_2^{\kappa}$. ρ is an invertible linear transformation over $2n$ bits. For simplicity, we consider only the case where A, M, C consist of full blocks, i.e. their size is a multiple of n, so that we can neglect the domain separation control logic, padding and length extension attacks which are irrelevant to our analysis. T is the authentication tag. S_i, S_o are the input and output states of the underlying cipher,

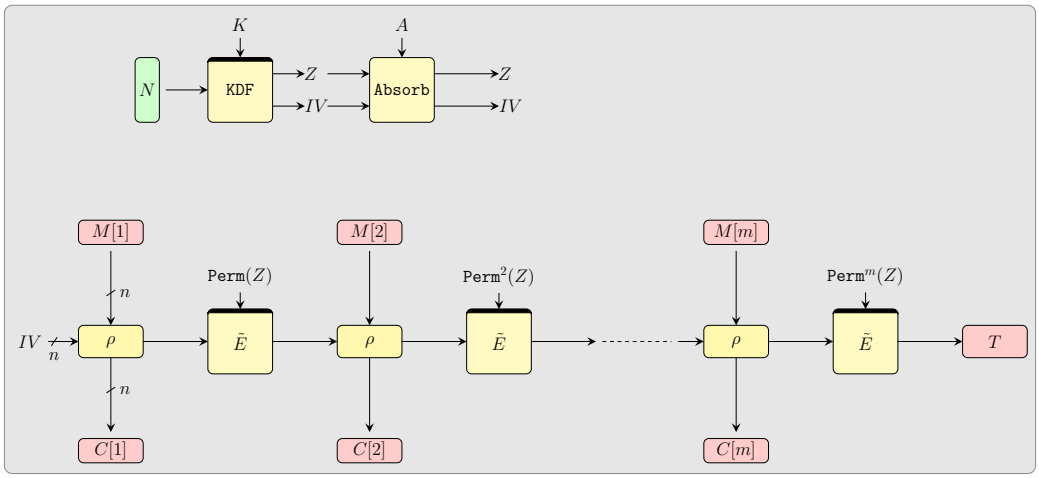

Figure 2: The RaC representation of COFB-like modes

respectively. KDF is the key derivation function, which is what distinguishes modes that follow the RaC representation. An RaC algorithm consists of three phases

1. (Z, IV) =KDF(N, K)
2. (Z, IV) =Absorb(A, Z, IV)
3. (C, T) =Enc(M, Z, IV)

which are elaborated in Figure 2. Since it is straightforward to see how each of the modes discussed in the paper is an instantiation of RaC, we will not go into the details of these instantiations, unless such information is necessary for our analysis. In order for the attacks in this paper to work, RaC must satisfy the following properties:

1. Given A, Absorb is invertible in terms of Z and IV.
2. ρ is linear and invertible.

3.1 Generic Attacks against RaC

Similar to any AEAD scheme, the schemes based on RaC are vulnerable to a wide range of generic attacks that can apply to any AEAD scheme. In order to provide a reference point for the reader to compare our attacks to these generic attacks, we provide a non-exhaustive list of examples in Table 2.

3.2 Forgery attacks against RaC

If Z is a fixed-point of Perm, RaC is vulnerable to forgery attacks. For example, assume that M consists of two blocks. The adversary can apply the following attack:

1. Ask for the encryption of M.
2. Calculate the internal state values for S_i, S_o at different points in the execution.
3. Find a ciphertext block C_x and a plaintext block M_x such that $(S_i[1], C_x) = \rho(S_o[0], M_x)$. This is very easy due to the properties of ρ.

Table 2: Some generic attacks against RaC. $H(x)$ is the information theoretic entropy of x. We assume that Z and IV are uniformly random. In practice the birthday attack costs may be different if this assumption is not valid.

Attack	Complexity
Master Key Guessing	$\mathcal{O}(2^{\lvert K\rvert})$
Tag Guessing	$\mathcal{O}(2^{\lvert T\rvert})$
State Guessing	$\mathcal{O}(2^{H(Z)+H(IV)})$
State Matching (Birthday Attacks)	$\mathcal{O}(2^{\frac{H(Z)+H(IV)}{2}})$
Encryption-Decryption Collision	$\mathcal{O}(2^{\frac{H(Z)+H(IV)}{2}})$
Online-Offline Collision	$\mathcal{O}(2^{\frac{H(Z)+H(IV)}{2}})$

4. Use the same tag from the encryption query with the ciphertext C_x to build a verification query.

If Z is not a fixed-point, but a member of a short cycle of period l, then the attack is modified to

1. Ask for the encryption of M of length $2l$.
2. Calculate the internal state values for S_i, S_o at different points in the execution.
3. Find a ciphertext block C_x and a plaintext block M_x such that $(S_i[l], C_x) = \rho(S_o[0], M_x)$. This is very easy due to the properties of ρ.
4. Use the same tag from the encryption query with the ciphertext $C_X = C_x|C[l+1]|C[l+2]|\cdots|C[2l-1]$ to build a verification query.

Another possible forgery attack is

1. Ask for the encryption of M of length $l+1$.
2. Calculate the internal state values for S_i, S_o at different points in the execution.
3. Find a ciphertext block C_x and a plaintext block M_x such that $(S_i[0], C_x) = \rho(S_o[l], M_x)$. This is very easy due to the properties of ρ.
4. Use the same tag from the encryption query with the ciphertext $C_X = C[0]|C[1]|\cdots|C[l-1]|C_x|C[1]|C[2]|\cdots|C[l]$ to build a verification query.

However, since Z is secret, it is not straightforward. Let p_l be the probability that a key Z picked uniformally at random is a member of a cycle of period l, then the attacks described above have a success probability of p_l and data complexity $2l$ blocks for an adversary who assumes Z is a member of such cycle. In order to have a success probability close to 1, the adversary requires roughly a number of encryption/decryption queries $q = p_l^{-1}$, and data complexity of $2ql$ blocks. A very important observation is that this attack does not depend on K, N, A or IV. Hence, an adversary targeting μ users simultaneously can rearrange his resources, spending only $q_k = \frac{q}{\mu}$ queries per key, expecting to achieve at least 1 successful forgery. The adversarial advantages of these attacks are qp_l in the single-key setting and μqp_l in the multi-key setting. Alternatively, the number of blocks required for these attacks is roughly $\sigma = 2ql$ and the advantages are $\frac{\sigma p_l}{2l}$ and $\frac{\mu\sigma p_l}{2l}$ for the single- and multi-key settings, respectively.

In order for a scheme to be immune to this attack, the designer needs to make sure that $\frac{2l}{p_l}$ is smaller than the complexities of the generic attacks in Table 2, or at least small enough to fall within the targeted security level for every possible value of l.

3.3 Analysis of COFB, HyENA and Remus

COFB and HyENA are very similar, with the main difference is in the ρ function. However, in both cases they satisfies our requirements. Hence, the same analysis can be applied to both modes. In this case, the outputs of the KDF are $Z = K|L$ and $IV = L|R$, where L and R are $n/2$-bit random variables, K is the k-bit master key, where $k = \kappa - n/2$, and $\cdot|\cdot$ represents the concatenation of two binary strings. $\texttt{Perm}(K|L) = K|\alpha \cdot L$, where α is a primitive element of $\mathrm{GF}(2^{n/2})$. If $L = 0$, the attack we described can apply. In other words, Perm has 2^k fixed-points and the probability of Z being one of them is $2^{-n/2}$. In the single key setting, this leads to an attack with complexity roughly $2^{n/2}$. However, the designers of COFB and HyENA do not claim security beyond $2^{n/2-log(n)}$ online queries with negligible number of offline queries, which means that this attack does not pose a threat against them. In the multi-key setting, the attack is more problematic, since it only requires $\frac{2^{n/2}}{\mu}$ data complexity per key. In scenarios where the number of keys used is relatively high, e.g. 2^{24}, these modes offer only 40-bit security against forgery over all keys, i.e. with 2^{40} queries per key, one of the users can be vulnerable to forgery. As we will describe later, these results are very similar to the results against COMET. However, a characteristic difference is that this only applies to forgery in the case of COFB and HyENA, while all the security of COMET breaks down, as we will see later.

In the case of Remus-N1, the initial value of Z is an n-bit random value, while $\texttt{Perm}(Z) = 2 \cdot Z$ over $\mathrm{GF}(2^n)$. Hence, there is only one possible fixed point at $Z = 0$ with probability 2^{-n}. The attack complexity in the single-key setting is roughly 2^n, with advantage $\frac{q}{2^n}$. This attack is matching to one of the bounds given by the designers in the single key setting. Moreover, the designers of Remus-N1 claim only $n/2$ bit security. In the multi-key setting, the situation is better than COFB and HyENA, where even with μ users, the scheme will still be immune to this attack as long as $\mu \leq 2^{n/2}$.

Last but not least, in the case of Remus-N2, the initial value of $Z = L|V$ is a $2n$-bit random value, while $\texttt{Perm}(Z = L|V) = 2 \cdot L|2 \cdot V$ over $\mathrm{GF}(2^n)$. In this case, there are three classes of weak keys:

1. $L = 0, V \neq 0$: In this case, the scheme can be reduced to an Even-Mansour-like cipher with key V. If this construction is secure then the security should be the same as Remus-N1, i.e. the scheme would still enjoy $n/2$-bit security. Since this event happens with probability 2^{-n}, then an attack based on this class of weak keys is much more costly than the claimed n-bit security.

2. $L \neq 0, V = 0$: The scheme is reduced to Remus-N1. Since this event happens with probability 2^{-n}, then an attack based on this class of weak keys is much more costly than the claimed n-bit security.

3. $L = 0, V = 0$: If this happens the forgery is successful with probability 1. However, the probability of such weak key is only 2^{-2n}, which means that after 2^n online queries, the success probability of forgery is 2^{-n}.

Similar to Remus-N1, the security degradation of Remus-N2 with regards to our attack is too slow to pose any practical threat. However, aside from the weak key analysis, it is clear that for Remus will have at least one bound on the form of $\frac{\mu q_o q_f}{2^{|Z|}}$ due to Online-Offline matching, since Z is random, where q_o is the number of online query blocks and q_f is the number of offline query blocks, using the folklore result mentioned by Luykx et al. [LMP17]. An open question is to prove multi-key security bounds for Romulus, which can also be modeled using RaC without any weak keys at all, i.e. has potential to satisfy the condition from Luykx et al. [LMP17].

4 Application to COMET-128

4.1 Brief Description of COMET-128

COMET-128 [GJN19a] is a block cipher based AEAD algorithm submitted to the NIST Standardization Process for Lightweight Cryptography. It can be described using RaC with the initial value of $Z = L|R$ is a random n-bit value and $IV = K$. During the third phase of RaC, $\texttt{Perm}(Z) = 2 \cdot L|R$ over $\mathrm{GF}(2^{\kappa/2})$. We show that for every pair $(N|A, K)$ there are $2^{\kappa/2}$ weak keys, with probability $2^{-\kappa/2}$ that the KDF outputs one of these keys. The existence of these weak keys and the applicability of the multi-key analysis leads to a set of interesting results:

1. After one query of length at least 32 bytes, forgery is successful with adversarial advantage 2^{-64}.
2. With 2^{63} online queries of 32 bytes each, forgery is successful with adversarial advantage $\sim 2^{-1}$.
3. With 2^{64} online queries of 32 bytes each, forgery is successful with adversarial advantage approaching ~ 1.
4. If the forgery is successful, the master key is identified easily with high probability with 2^{65} offline queries.

4.2 Existence of Weak Keys

The specification of COMET-128 does not include any discussion on the weak key analysis. COMET-128 is an instance of RaC, where the master key is used as an IV, and the session key $Z = L|R$ is used as the block cipher key. If the value of $L = 0^{64}$, then Z is constant over all the blocks. Since this event is defined over 64 bits of Z, which is chosen uniformally at random using the KDF (a permutation over $\mathrm{GF}(2^{128})$), there are 2^{64} weak values of Z for each stage of the algorithm and the probability that Z at a certain stage is weak is equal to $\frac{2^{64}}{2^{128}} = 2^{-64}$. Since COMET-128 applies the KDF with a different N for every different message and since the KDF is a permutation, given μ online queries, we get μ messages encrypted with μ different values Z_i.

4.3 Existential Forgery Attack with Weak Keys

Given we have established how the weak keys behave and their probability, we describe how to forge a ciphertext once a weak key has been sampled by the KDF. Let M be the known message encrypted with a weak key $0^{64}|R$, where $|M| \geq 256$. Let M_1 and M_2 be the first two message blocks after parsing M, with C_1 and C_2 as the corresponding ciphertext blocks. Since the attacker knows M and C, he can retrieve the internal state values S_1 and S_2, where S_1 is the state before the absorption of M_1 and S_2 is the state after the absorption of M_2. Hence, we have

$$S_1 = \mathrm{Shuffle}^{-1}(M_1 \oplus C_1) \tag{3}$$

and

$$S_2 = M_2 \oplus \mathrm{Shuffle}^{-1}(M_2 \oplus C_2) \tag{4}$$

The attacker wants to find M_x and C_x, such that

$$S_1 = \mathrm{Shuffle}^{-1}(M_x \oplus C_x) \tag{5}$$

and

$$S_2 = M_x \oplus \text{Shuffle}^{-1}(M_x \oplus C_x) \tag{6}$$

which is a simple well-defined linear system of equations defined over 256 Boolean variables and easily solvable. The attacker removes C_1 and C_2 from the ciphertext, and inserts C_x in the location of C_1, while shifting the rest of the ciphertext 16 bytes backwards and reducing the ciphertext size by 16, leading to a successful forgery. This attack has been verified by modifying the reference implementation of COMET-128 to use some weak keys Z_i. The overall complexity of the attack is 2^{64} online queries, 0 offline queries and succeeds with probability close to 1.

4.4 Key Recovery Attack

The previous existential forgery attack can be used as a filter to discover the occurrence of a weak key. Once the forgery succeeds, we know that Z during the message encryption phase of the algorithm has one of the weak key values, which are 2^{64} values. The attacker can then choose a message that has been previously encrypted with a weak key, and reverse the algorithm with each of these values. Since the master key K is used as an IV in COMET-128, this will lead to 2^{64} possible key candidates. For each of these key candidates, the attacker can apply $\texttt{KDF}(N, K)$ and verify whether the KDF generates the corresponding Z. Since the probability that $E_K(N) = Z$ is 2^{-128}, we expect to be able to uniquely identify the master key at this point, which completely breaks the system. The complexity is 2^{65} offline block cipher queries. This attack is given in details in Figure 3, where COMETE and COMETD represent online queries to the encryption and decryption oracles, respectively, forge(C) represents the forgery attack in Section 4.3, $\mathcal{Z}_{\text{weak}}$ is the set of weak keys of COMET with empty AD and $\text{AESE}_K/\text{AESD}_K$ are the offline primitive queries for AES encryption and decryption, respectively. $\xleftarrow{\$}$ represents random sampling without replacement, in order to respect the nonce model, so N and M are never repeated. $\perp$ represents failed decryption. The first loop of the algorithm (lines 2-8) requires at most 5 blocks of storage, or 80 bytes, and is expected to run 2^{64} times, with 2^{65} data complexity. The second loop (lines 10-18) runs exactly 2^{64} times, with two primitive calls for each iteration, and requires 80 bytes of storage, in addition to the successful key candidates. Due to the properties of AES as a secure block cipher, it is expected that only 1 candidate is successful.

4.5 Summary of Results

The designers claim that the `permute` function has a period of 2^{64} and hence the only way for the key to repeat is to encrypt more than 2^{64} blocks. We have shown that this is not correct, since there are 2^{64} keys with period 1 and consequently this is another way for the key to repeat, and it is easily detectable with a single verification query. The designers also base their security analysis on a set of generic attacks on the scheme. We interpret them as suggested by the designers in section 5.1.4 of [GJN19a]. When substituting in the bounds they give for the values of n = the block size and κ = the key size, we get the results in Table 3 for integrity and Table 4 for privacy. Our weak key analysis shows has significantly lower complexity in all cases.

4.6 Attacks in the Multi-Key Setting

Combining the previous two attacks with the multi-key analysis shows a security concern about the design of COMET-128. While this flaw does not violate the 64-bit security and NIST requirements in the single key setting, in practice it can lead to drastic results.

$M_x \leftarrow \perp$
while $M_x = \perp$ do
  $M \xleftarrow{\$} \{0,1\}^{2n}$
  $N \xleftarrow{\$} \{0,1\}^{n}$
  $(C,T) \leftarrow \mathsf{COMETE}(M,N)$
  $C_x \leftarrow \text{forge}(C)$
  $M_x \leftarrow \mathsf{COMETD}(C_x,T,N)$
endwhile
$i = 0$
for $Z \in \mathcal{Z}_{\text{weak}}$ do
  $S \leftarrow \text{Shuffle}^{-1}(M_x \oplus C_x)$
  $IV \leftarrow \mathsf{AESD}_Z(S)$
  $Y \leftarrow \mathsf{AESE}_{IV}(N)$
  if $Y = Z$ do
    $K_i \leftarrow IV$
    $i = i + 1$
  endif
endfor
return $\{K_j | 0 \leq j < i\}$

Figure 3: Full key recovery attack in the single-key setting

Table 3: Integrity Claims Made by the Designers of COMET-128 vs. Our Attack: D_e is the number of encryption queries, D_v is the number of verification queries and T is the number of offline primitive queries

Method	D_e	D_v	T
Tag Guessing [GJN19a]	-	2^{128}	-
Decryption-Encryption Matching [GJN19a]	2^{121}	2^{128}	-
Decryption-Offline Matching 1 [GJN19a]	-	2^{121}	2^{128}
Decryption-Offline Matching 2 [GJN19a]	-	2^{65}	$2^{122.5}$
Weak Key Analysis [Ours]	2^{65}	2^{64}	-

Similar to COFB and HyENA, given μ users, the per-user forgery security is reduced to $64 - log(\mu)$ bits. For example, given 4 million users, forgery is successful against at least 1 user with close to 1 probability given 2^{40} queries per user. Once the forgery succeeded against at least 1 user, the corresponding key can be recovered using 2^{65} offline encryptions. Recovering as many as one master key using 2^{40} queries per user and 2^{65} offline encryptions cannot be considered impractical and can lead to practical attacks, as it is well within the practical limits set by the NIST for each individual user and the attacker can still use this key $2^{50} - 2^{40}$ times to impersonate the fallen user or to eavesdrop on the communications. This attack is given in details in Figure 4. The notation is the same as Figure 3, except U is the id of the fallen user, S_μ is the set of targeted users such that $\mu = |S_\mu|$ and $\mathsf{COMETE}_u/\mathsf{COMETD}_u$ are online queries for a specific user u. The offline resources are the same as the single key attack, while the online queries per user are $\mathcal{O}(\frac{2^{64}}{\mu})$.

4.7 Simple Fix

We propose two potential simple fixes that have the potential of eliminating the problems we discussed:

1. Eliminate the weak keys completely by replacing the doubling with an arithmetic

Table 4: Privacy Claims Made by the Designers of COMET-128 vs. Our Attack: D_e is the number of encryption queries, D_v is the number of verification queries and T is the number of offline primitive queries

Method	D_e	D_v	T
Online-Online Matching [GJN19a]	2^{65}	-	2^{191}
Online-Offline Matching [GJN19a]	2^{65}	-	2^{183}
Key Guessing [GJN19a]	1	-	2^{128}
Weak Key Analysis [Ours]	2^{65}	2^{64}	2^{65}

$M_x \leftarrow \perp$
while $M_x = \perp$ do
  for $u \in S_\mu$ do
    if $M_x = \perp$ do
      $M \xleftarrow{\$} \{0,1\}^{2n}$
      $N \xleftarrow{\$} \{0,1\}^{n}$
      $(C,T) \leftarrow \mathsf{COMETE}_u(M,N)$
      $C_x \leftarrow \text{forge}(C)$
      $M_x \leftarrow \mathsf{COMETD}_u(C_x,T,N)$
      if $M_x \neq \perp$ then $U \leftarrow u$
    endif
  endfor
endwhile
$i = 0$
for $Z \in \mathcal{Z}_{\text{weak}}$ do
  $S \leftarrow \text{Shuffle}^{-1}(M_x \oplus C_x)$
  $IV \leftarrow \mathsf{AESD}_Z(S)$
  $Y \leftarrow \mathsf{AESE}_{IV}(N)$
  if $Y = Z$ do
    $K_i \leftarrow IV$
    $i = i + 1$
  endif
endfor
return $(U, \{K_j | 0 \leq j < i\})$

Figure 4: Full key recovery attack in the multi-key setting

counter. However, arithmetic counters are more costly in hardware implementations. A 64-bit arithmetic counter can be very slow.

2. Use doubling over a larger field, e.g. GF(2^{128}), in order to reduce the number and probability of weak keys.

5 Application to mixFeed

5.1 Brief Description of mixFeed

mixFeed [CN19a] is an AES-based AEAD algorithm submitted to round 2 of the NIST Lightweight Cryptography Standardization Process. It uses a hybrid feedback structure, where half the input to the block cipher comes directly from the plaintext, while the other half is generated from the previous block cipher call and the plaintext in a CBC-like manner. The initial session key is generated using a KDF that depends on the master key

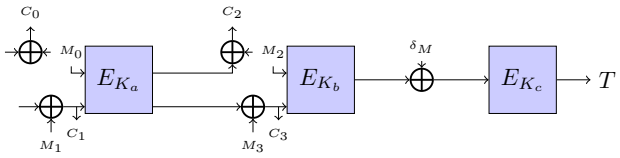

Figure 5: Part of the Encryption Phase of mixFeed

K and the nonce N, then each block key is the output of applying a permutation P to the previous block key. The permutation P is defined as 11 rounds of the AES key Scheduling Algorithm [DR13]. The encryption part is shown in Figure 5. It was shown in a previous attack [Kha19] that given a known plaintext/ciphertext pair, the attacker can force the input to the block cipher to a certain value during the forgery challenge, so we consider this part of the analysis as a given and we do not include it in this article. Forcing the input to the block cipher is not enough to lead to an attack and is applicable to many block cipher based schemes. The question is whether the attacker can get enough information about the session key to increase the adversarial advantage.

5.2 Weak Key Analysis of mixFeed

The designers of mixFeed discuss the multi-key analysis in a brief statement in the specification. However, they do not mention the weak-key analysis. At first, it is not obvious why the weak key analysis is relevant to mixFeed. However, when we study how the mode operates, it is quite similar to modes like COMET, except that the key update function between blocks is not a multiplication by constant over a finite field, but it is the key schedule permutation of AES itself. In other words, every block cipher call takes as a key $K_i = P(K_{i-1})$, where K_{i-1} is the key used in the previous block and P is the permutation that applies 11 rounds of the AES key schedule. As explained in Section 3.2, different types of forgery succeed if the key is repeated, i.e. if the permutation cycle used to update the key is smaller than the message length. If the permutation is well designed, e.g. maximal length LFSR or arithmetic counter, the probability of this event should be very low. Also, if the permutation is an ideal permutation picked uniformally at random, it should have n cycles whose lengths follow a Poisson distribution.

The AES key schedule permutation is not designed to be an ideal permutation and it should not be used as one. It can be described as a permutation over four 32-bit words, which consists of 11 rounds. The rounds differ only in the round constant. We define the permutation f_c over a 32 bit word as the feedback function in round c as:

$$W \rightarrow \text{SubWord}(W \ggg 8) + \text{rcon}(c) \tag{7}$$

where $W \ggg r$ represents bitwise right rotation of W by r bits and $\text{rcon}(c)$ is defined as $x^{c\%11}|0^{24}$ such that x is defined over $\text{GF}(2^8)$. Given this permutation, a single round of the AES key schedule can be defined as

$$W_0, W_1, W_2, W_3 \rightarrow W_0 \oplus f_c, W_1 \oplus W_0 \oplus f_c, W_2 \oplus W_1 \oplus W_0 \oplus f_c, W_3 \oplus W_2 \oplus W_1 \oplus W_0 \oplus f_c \tag{8}$$

where f_c is applied to W_3 and eight unrolled rounds can be defined as in Table 5.

In fact, there is an iterative structure over 4 rounds, where we can write the value of any key word after 4 rounds in terms of the initial value of this word and a certain set of feedback functions. If a key is a fixed point over R rounds, where R is a multiple of 4, then the involved feedback functions must add up to 0. If the feedback function is ideal, we expect this to happen with probability 2^{-32} for each word and 2^{-128} in total, but of

Table 5: 8 round unrolling of the AES key schedule

Round 0	W_0	W_1	W_2	W_3
Round 1	$W_0 \oplus f_0$	$W_1 \oplus W_0 \oplus f_0$	$W_2 \oplus W_1 \oplus W_0 \oplus f_0$	$W_3 \oplus W_2 \oplus W_1 \oplus W_0 \oplus f_0$
Round 2	$W_0 \oplus f_0 \oplus f_1$	$W_1 \oplus f_1$	$W_2 \oplus W_0 \oplus f_0 \oplus f_1$	$W_3 \oplus W_1 \oplus f_1$
Round 3	$W_0 \oplus f_0 \oplus f_1 \oplus f_2$	$W_1 \oplus W_0 \oplus f_0 \oplus f_2$	$W_2 \oplus W_1 \oplus f_1 \oplus f_2$	$W_3 \oplus W_2 \oplus f_2$
Round 4	$W_0 \oplus f_0 \oplus f_1 \oplus f_2 \oplus f_3$	$W_1 \oplus f_1 \oplus f_3$	$W_2 \oplus f_2 \oplus f_3$	$W_3 \oplus f_3$
Round 8	$W_0 \oplus \bigoplus_{i=0}^{7} f_i$	$W_1 \oplus \bigoplus_{i=0}^{3} f_{2*i+1}$	$W_2 \oplus f_2 \oplus f_3 \oplus f_6 \oplus f_7$	$W_3 \oplus f_3 \oplus f_7$

course, this is not the case. It is trivial to see that there is only 1 value which is a fixed point after 4 rounds, and in general the conditions for a fixed point after R rounds are

$$\bigoplus_{i=0}^{R/4-1} f_{3+4i} = 0 \tag{9}$$

$$\bigoplus_{i=0}^{R/4-1} f_{2+4i} = 0 \tag{10}$$

$$\bigoplus_{i=0}^{R/4-1} f_{1+4i} = 0 \tag{11}$$

$$\bigoplus_{i=0}^{R/4-1} f_{4i} = 0 \tag{12}$$

$$\tag{13}$$

For example, fixed points over 8 rounds must satisfy

$$f_3 \oplus f_7 = 0 \tag{14}$$

$$f_2 \oplus f_6 = 0 \tag{15}$$

$$f_1 \oplus f_5 = 0 \tag{16}$$

$$f_0 \oplus f_4 = 0 \tag{17}$$

$$\tag{18}$$

The last condition can be written as $f_0(W_3) \oplus f_4(W_3 \oplus f_3) = 0$. Since f_0 and f_4 differ only in the constant value, we can rewrite the condition as

$$\text{SubWord}(W \ggg 8) \oplus \text{SubWord}(W \ggg 8 \oplus \Delta) = \delta \tag{19}$$

where $\Delta = f_3$ and $\delta = \text{rcon}(4) \oplus \text{rcon}(0)$. Clearly, this a non-linear equation. What is interesting, is that this equation is defined over the Sbox of AES and can be divided into three equations on the form $Sbox(x) \oplus Sbox(x \oplus y) = 0$ and one equation of the form $Sbox(x) \oplus Sbox(x \oplus y) = a$. For the first three equations, $y = 0$ since the AES Sbox is bijective, while for the last one y has 127 possible values that can be retrieved from the Difference Distribution Table of the AES Sbox. Hence, we reduce the possibilities of f_3 to 127 values. Then,

$$f_3(W_2 \oplus W_3 \oplus f_2) \oplus f_7(W_2 \oplus W_3 \oplus f_2 \oplus f_6) = f_3(W_2 \oplus W_3 \oplus f_2) \oplus f_7(W_2 \oplus W_3) = 0 \tag{20}$$

Hence, similar arguments can be made about f_2 and similarly f_1 and f_0. By such argument, one expects roughly about $2^{27.9}$ fixed points for the reduced-round AES key

Table 6: Representatives of 20 Cycles of length= 1133759136 for the AES Key Schedule 11 Round Permutation Used in mixFeed

000102030405060708090a0b0c0d0e0f
00020406080a0c0e10121416181a1c1e
0004080c1014181c2024282c3034383c
000810182028303840485058606870 78
001020304050607080 90a0b0c0d0e0f0
10111213141516171819 1a1b1c1d1e1f
20222426282a2c2e30323436383a3c3e
4044484c5054585c6064686c7074787c
80889098a0a8b0b8c0c8d0d8e0e8f0f8
303132333435363738393a3b3c3d3e3f
707172737475767778797a7b7c7d7e7f
000306090c0f1215181b1e2124272a2d
00050a0f14191e23282d32373c41464b
00070e151c232a31383f464d545b6269
000d1a2734414e5b6875828f9ca9b6c3
00152a3f54697e93a8bdd2e7fc11263b
00172e455c738aa1b8cfe6fd142b4259
00183048607890a8c0d8f00820385068
001c3854708ca8c4e0fc1834506c88a4
001f3e5d7c9bbad9f81736557493b2d1

schedule of 8 rounds. One can go analyzing more rounds. While this problem is interesting on its own regard, it is not the scope of our result and we leave it to future work. We just mention the analysis to show that the AES Key Schedule is far from an ideal permutation and also because the cycle length we have found is a multiple of 4.

We have run a simple cycle finding script using brute force and found at least 20 cycles of length $1133759136 \sim 2^{30.08}$, out of 33 seeds we have tried. We give a representative of each of those cycles in Table 6, in case the reader want to verify the results. We will also make our simple script available online. It is not clear to us why this number is special. However, this means that there are at least $2^{34.4}$ weak keys which allow forgery of messages of length $2^{30.08}+1$ blocks into messages of length $2^{31.08}+1$. Finding each of these cycles takes around 1 hour on a single-core personal computer using brute force, hence we do not know how many cycles are there of these structure or of different length. We found a large set of values that do not belong to cycles of that length or smaller. Our findings are good enough to show that the security bounds claimed by the designers of mixFeed are not true and it shows a gap in their analysis. The designers claim that the adversarial advantage is $\frac{\sigma T}{2^{192}}$. However, as we explained in the analysis of COMET-128, by applying multi-key-weak-key analysis, we see that the advantage increases drastically. According to the designers, after encrypting a message of $2^{30.08}+1$ blocks and decrypting a message of length $2^{31.08}+1$ blocks, the adversarial advantage should be $2^{-160.92}$. However, the weak key analysis show that forgery is successful with probability *at least* $2^{-93.59}$. After encrypting $2^{18.92}$ such messages, with overall online complexity of 2^{50} blocks, the adversarial advantage is $2^{-74.68}$.

While this result does not make mixFeed insecure, it shows a huge gap in the security analysis of mixFeed and calls for further cryptanalytic efforts if mixFeed is to be used in the real world. As mentioned earlier, our result is just a lower bound found by brute forcing some key values. For example, there is a big question mark on the special cycle length we found, which may potentially be related to more cycles, further increasing the advantage. Since we cannot do a full characterization of the AES Key Schedule Permutation

to find all the cycles and given that the experiments show that a certain cycle length is highly probable, we rely on the statistical result we found to conclude that a cycle length of 1133759136 has high probability. None of these results are conclusive. Nevertheless, without proper understanding of the underlying permutation, and given our experimental results which show a huge gap between the security claims and reality, it is hard to argue for the security of mixFeed. Similar to COMET, COFB, HyENA and Remus, the forgery attack against mixFeed is vulnerable to multi-key security degradation. It is worth noting that, unlike in the case of COMET, the weak key analysis does not lead to the master key recovery, since mixFeed does not use the master key during the main part of the encryption.

5.3 Summary of Results

The designers of mixFeed make a security claimed in their round 1 submission that the adversarial advantage is bounded by $\frac{DT}{2^{192}}$. We believe a better characterization would have been $max(\frac{DT}{2^{192}}, \frac{T}{2^{128}}, \frac{q}{2^{128}})$, where D is the total number of encrypted/decrypted blocks, T is the total number of offline primitive calls, and q is the total number of encryption queries, since the designers should have included key guessing and tag guessing attacks. We have shown that the integrity bound for the adversarial advantage cannot be lower than 2^{-75} at $(q, D, T) = (2^{18.92}, 2^{50}, 1)$, a lot higher than any of the three bounds mentioned before, with the results making it very unconvincing that this is a tight lower bound.

6 Conclusion

We have applied our analysis to two AEAD modes: COMET-128 and mixFeed, showing gaps in their security analysis. In the case of COMET-128, the whole security of the system breaks when the number of queries approaches the bound of 2^{64}. Moreover, by targeting a small number of users simultaneously, one of the user keys can be retrieved with low per-user complexity. This is specific to COMET-128 and not a generic attack on similar modes as it uses a permutation with short cycles and many fixed points and uses the master key as part of the plaintext input (as the IV to be exact). We also propose two simple fixes. In the case of mixFeed, we enhanced the adversarial advantage by a huge factor compared to the designers claim, where with only $2^{18.92}$ guesses he has *at least* an advantage of $2^{-74.68}$. The attack in this case does not contradict the claims made by the designers but it shows a gap in their analysis that needs to be addressed. We do not see how can this be fixed for mixFeed, since the choice of P is inherit in the mode design and even if instantiated with a different cipher, we expect the security degradation to be even more drastic, as AES has a strong key scheduling permutation compared to other ciphers. However, whether it can be further enhanced is inconclusive. In the multi-key setting, we showed limitations of COFB, Remus and HyENA and a weakness of COMET-128. Part of the future work is to derive tight bounds for RaC-based modes in the multi-key setting.

Acknowledgments

I would like to thank the anonymous reviewers of ToSC for their constructive comments. I would like to thank Thomas Peyrin, Tetsu Iwata and Kazuhiko Minematsu on many fruitful discussions on topics related to this analysis, including weak keys, birthday bound security and the AES Key Scheduling. I would like to thank Mridul Nandi, Shay Gueron, Ashwin Jha and Bishwajit Chakraborty for reading and discussing the early version of this article.

References

[ABT14] Mohamed Ahmed Abdelraheem, Andrey Bogdanov, and Elmar Tischhauser. Weak-Key Analysis of POET. *IACR Cryptology ePrint Archive*, 2014:226, 2014. https://eprint.iacr.org/2014/226.pdf.

[BCI+19] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Annual International Cryptology Conference*, pages 123–153. Springer, 2016. https://link.springer.com/chapter/10.1007/978-3-662-53008-5_5.

[BT16] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In *Annual International Cryptology Conference*, pages 247–276. Springer, 2016. https://link.springer.com/chapter/10.1007/978-3-662-53018-4_10.

[CDJN19] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, and Mridul Nandi. HyENA. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[CIMN17] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 277–298. Springer, 2017. https://link.springer.com/chapter/10.1007/978-3-319-66787-4_14.

[CN19a] Bishwajit Chakraborty and Mridul Nandi. mixFeed. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[CN19b] Bishwajit Chakraborty and Mridul Nandi. Security Proof of mixFeed. NIST Lightweight Cryptography Workshop, 2019. https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2019/documents/papers/security-proof-of-mixfeed-lwc2019.pdf.

[DR13] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[GJN19a] Shay Gueron, Ashwin Jha, and Mridul Nandi. COMET: COunter Mode Encryption with authentication Tag. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[GJN19b] Shay Gueron, Ashwin Jha, and Mridul Nandi. On the Security of COMET Authenticated Encryption Scheme. NIST Lightweight Cryptography Workshop, 2019. https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2019/documents/papers/on-sthe-security-of-comet-lwc2019.pdf.

[HP08] Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based MAC algorithms. In *Annual International Cryptology Conference*, pages 144–161. Springer, 2008. https://link.springer.com/chapter/10.1007/978-3-540-85174-5_9.

[IKMP19a] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the titans: The romulus and remus families of lightweight aead algorithms. Cryptology ePrint Archive, Report 2019/992, 2019. https://eprint.iacr.org/2019/992.

[IKMP19b] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. REMUS. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[IKMP19c] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[Kha19] Mustafa Khairallah. Forgery Attack on mixFeed in the Nonce-Misuse Scenario. *IACR Cryptology ePrint Archive*, 2019:457, 2019. https://eprint.iacr.org/2019/457.pdf.

[LMP17] Atul Luykx, Bart Mennink, and Kenneth G Paterson. Analyzing multi-key security degradation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 575–605. Springer, 2017. https://link.springer.com/chapter/10.1007/978-3-319-70697-9_20.

[nis18] Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process, 2018. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf.